Тысяча и один бэкап

Резервное копирование Compute Cloud

Артемий Капитула
Mail.Ru Cloud Solutions





А нужно ли резервное копирование

Резервирование – защита от отказа

Резервное копирование – защита от ошибок



Резервное копирование

Стадии процесса и нагружаемые компоненты

Прочесть - диск и сеть

Обработать и сжать - процессор и память

Записать – диск и сеть



Резервное копирование

Стадии процесса и нагружаемые компоненты

Прочесть - диск и сеть

Обработать и сжать - процессор и память

Записать – диск и сеть



Резервное копирование

Стадии процесса и нагружаемые компоненты

Прочесть - диск и сеть

Обработать и сжать - процессор и память

Записать – диск и сеть



Клиенты Compute Cloud

Cloud-native / cloud-ready

Stateless или state во внешнем сервисе

Используют резервирование и масштабирование

Используют PaaS

Classic/Legacy

Предполагают непрерывность вычислительного ресурса

Сохраняют state внутри VM

Нуждаются в резервном копировании

Используют IaaS



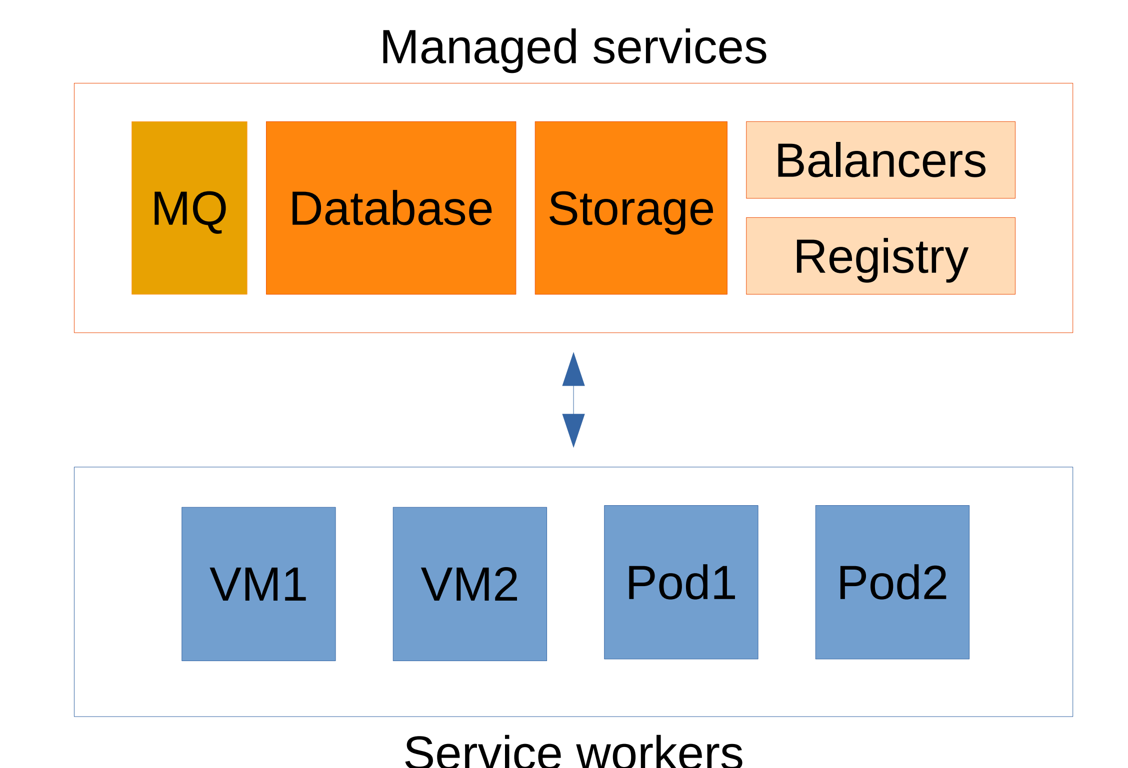
Клиенты Compute Cloud

Cloud-native

Stateless или state во внешнем сервисе

Используют резервирование и масштабирование

Используют PaaS





Клиенты Compute Cloud

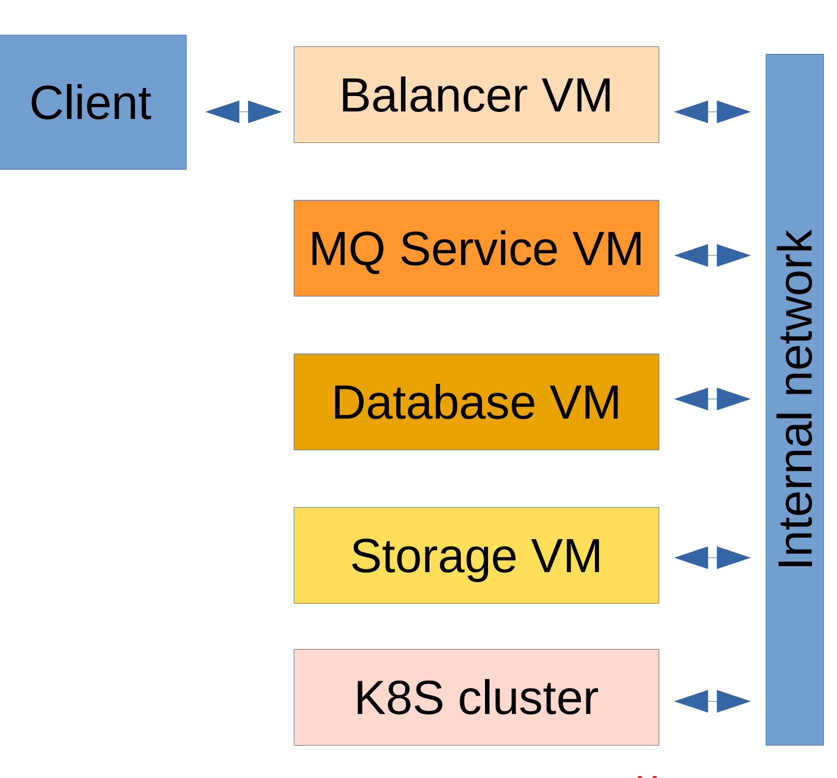
Classic/Legacy

Предполагают непрерывность вычислительного ресурса

Сохраняют state внутри VM

Нуждаются в резервном копировании

Используют IaaS





Резервные копии

Примерно 500TB в сутки (15PB в месяц +-)

350TB в течение 6 часов

Thin и Thick диски

Latency 0.1 … 15ms

Ceph, iSCSI, локальные диски

Offline/online диски

500TB декларативных

300 .. 350 TB реальных

Рост x2 в год

Требуется сжатие
* не для всех эффективно



Резервные копии

Примерно 500TB в сутки (15PB в месяц +-)

350TB в течение 6 часов

Thin и Thick диски

Latency 0.1 … 15ms

Ceph, iSCSI, локальные диски

Offline/online диски

Аффектятся:
* сеть
* диски

Никто не хочет замедления в рабочее время

Все хотят бэкапы “ночью”



Резервные копии

Примерно 500TB в сутки (15PB в месяц +-)

350TB в течение 6 часов

Thin и Thick диски

Latency 0.1 … 15ms

Ceph, iSCSI, локальные диски

Offline/online диски

Преаллоцированные:
быстрые
относительно дорогие

Тонкие:
Медленнее
Заметно более дешевые



Резервные копии

Примерно 500TB в сутки (15PB в месяц +-)

350TB в течение 6 часов

Thin и Thick диски

Latency 0.1 … 15ms

Ceph, iSCSI, локальные диски

Offline/online диски

Локальные SSD и NVMe
0.1 .. 0.2ms

Сетевые SSD и NVMe
0.3 .. 0.5ms

Ceph SSD
1 … 2ms

Ceph HDD
5 .. 15ms



Резервные копии

Примерно 500TB в сутки (15PB в месяц +-)

350TB в течение 6 часов

Thin и Thick диски

Latency 0.1 … 15ms

Ceph, iSCSI, локальные диски

Offline/online диски

iSCSI

* виден хосту
* использует блочный стек
* преаллоцирован

Ceph

* не виден хосту
* доступ через абстракции
* “тонкий”



Резервные копии

Примерно 500TB в сутки (15PB в месяц +-)

350TB в течение 6 часов

Thin и Thick диски

Latency 0.1 … 15ms

Ceph, iSCSI, локальные диски

Offline/online диски

Offline
* Отключенные
* Неизменяемые
* Консистентные
* Легко копировать
* Малый overhead

Online
* Подключенные
* Изменяющиеся
* Неконсистентные (?!)
* Нужны снапшоты
* Заметный overhead



Уложиться в окно времени

4TB диск

30MB/сек → ~140 тысяч секунд

Цель - 300MB/сек → ~14 тысяч секунд

Имеющиееся решение: 15 .. 50 MB/сек





Консистентность копии

Offline диск:

- Диск → Резервная копия



Консистентность копии

Offline диск:

Диск → Резервная копия

“Однодисковая” VM

Диск → Снапшот → Резервная копия



Консистентность копии

Offline диск:

Диск → Резервная копия

“Однодисковая” VM

Диск → Снапшот → Резервная копия

“Многодисковая” VM

Диск 1, Диск 2 → Снапшот 1, Снапшот 2 → Резервная копия



Консистентность копии

Offline диск:

Диск → Резервная копия

“Однодисковая” VM

Диск → Снапшот → Резервная копия

“Многодисковая” VM

Диск 1, Диск 2 → Снапшот 1, Снапшот 2 → Резервная копия



OpenStack backups

Karbor
менеджмент

Гипервизор
Nova / Compute

Backup API
Cinder / Backup

Гостевая ОС
Guest agent

СХД controlplane
Cinder / Volume

Cold storage
Своя реализация S3



Драйверы резервного копирования

Запрос → Frontend → Manager → Driver

Openstack Mainstream:

*** Фреймворк, модель in-process driver**
*** Однопоточный драйвер**
*** Greenlets**
*** Ужасный I/O**

Tivoli
Ceph
NFS / Posix FS
Gluster
Google

Non-mainstream legacy драйвер:

… есть поддержка своего S3
… а еще не работает L1

S3(v1 / v2)



Фреймворк

```
def backup(self, backup_info, volume_info):
    file_descriptor = self.open_volume(volume_info)
    try:
        self.driver.backup(backup_info, file_descriptor)
    finally:
        self.close_volume(volume_info, file_descriptor)
```



Не обязательно системный файловый дескриптор (прощай AIO)

Один дескриптор (один указатель текущей позиции, однопоточная работа)

Невозможно бэкапить быстрее чем читает один поток



Провальная итерация

Параллельное чтение

- **Два потока**
- **Основной читает из буфера**
- **Второй заполняет буфер**

15 – 50 MB/сек → 30 … 70 MB/сек

Укорение x2

Делайте вещи проще

… но не проще, чем они есть на самом деле

```
import zlib
import sys

with open(sys.argv[1], "rb") as f:
    cm = zlib.compressobj(-1, zlib.DEFLATED)
    while True:
        d = f.read(524288)
        if not d:
            break
        cm.compress(d)
```

```
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sdj               0.00     0.00  648.00    0.00 82944.00     0.00   256.00     0.64    0.99    0.99    0.00   0.54  34.90
sdj               0.00     0.00  684.00    0.00 87552.00     0.00   256.00     0.64    0.94    0.94    0.00   0.51  35.00
sdj               0.00     0.00  672.00    0.00 86016.00     0.00   256.00     0.65    0.98    0.98    0.00   0.52  35.10
sdj               0.00     0.00  676.00    0.00 86528.00     0.00   256.00     0.63    0.93    0.93    0.00   0.50  33.60
sdj               0.00     0.00  700.00    0.00 89600.00     0.00   256.00     0.68    0.97    0.97    0.00   0.53  36.80
sdj               0.00     0.00  684.00    0.00 87552.00     0.00   256.00     0.64    0.93    0.93    0.00   0.51  34.60
sdj               0.00     0.00  712.00    0.00 91136.00     0.00   256.00     0.65    0.92    0.92    0.00   0.50  35.50
```

/sys/block/sdj/queue/read_ahead_kb

Только I/O

```
import sys

with open(sys.argv[1], "rb") as f:
    cm = zlib.compressobj(-1, zlib.DEFLATED)
    while True:
        d = f.read(524288)
        if not d:
            break
```

Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	r_await	w_await	svctm	%util
sdj	0.00	0.00	2304.00	0.00	294912.00	0.00	256.00	1.81	0.79	0.79	0.00	0.43	99.40
sdj	0.00	0.00	2275.00	0.00	291200.00	0.00	256.00	1.80	0.79	0.79	0.00	0.44	99.30
sdj	0.00	0.00	2328.00	0.00	297984.00	0.00	256.00	1.78	0.77	0.77	0.00	0.43	99.60
sdj	0.00	0.00	2270.00	0.00	290560.00	0.00	256.00	1.79	0.79	0.79	0.00	0.44	98.90
sdj	0.00	0.00	2297.00	0.00	294016.00	0.00	256.00	1.79	0.78	0.78	0.00	0.43	99.40
sdj	0.00	0.00	2198.00	0.00	281344.00	0.00	256.00	1.76	0.80	0.80	0.00	0.45	98.40
sdj	0.00	0.00	2205.00	0.00	282240.00	0.00	256.00	1.73	0.78	0.78	0.00	0.44	96.80

Менять алгоритм сжатия?

Другой алгоритм сжатия

None (Read)	ZLIB	LZ4
~290 MB/s	~85 MB/s	~240 MB/s



Что можно сделать с I/O

```
import sys

with open(sys.argv[1], "rb") as f:
    cm = zlib.compressobj(-1, zlib.DEFLATED)
    while True:
        d = f.read(524288)
        if not d:
            break
```

```
import sys

with open(sys.argv[1], "rb") as f:
    cm = zlib.compressobj(-1, zlib.DEFLATED)
    while True:
        d = f.read(524288 * 8)
        if not d:
            break
```

```
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sdj               0.00     0.00 2304.00    0.00 294912.00     0.00   256.00     1.81    0.79    0.79    0.00   0.43  99.40
sdj               0.00     0.00 2275.00    0.00 291200.00     0.00   256.00     1.80    0.79    0.79    0.00   0.44  99.30
sdj               0.00     0.00 2328.00    0.00 297984.00     0.00   256.00     1.78    0.77    0.77    0.00   0.43  99.60

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
sdj               0.00     0.00 2313.00    0.00 296064.00     0.00   256.00     1.81    0.78    0.78    0.00   0.43  99.10
sdj               0.00     0.00 2269.00    0.00 290432.00     0.00   256.00     1.81    0.80    0.80    0.00   0.44  99.80
sdj               0.00     0.00 2215.00    0.00 283520.00     0.00   256.00     1.79    0.81    0.81    0.00   0.44  98.10
```



Что можно сделать с I/O

```
import sys

with open(sys.argv[1], "rb") as f:
    cm = zlib.compressobj(-1, zlib.DEFLATED)
    while True:
        d = f.read(524288)
        if not d:
            break
```

```
import sys

with open(sys.argv[1], "rb") as f:
    cm = zlib.compressobj(-1, zlib.DEFLATED)
    while True:
        d = f.read(524288 * 8)
        if not d:
            break
```

Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	r_await	w_await	svctm	%util
sdj	0.00	0.00	2304.00	0.00	294912.00	0.00	256.00	1.81	0.79	0.79	0.00	0.43	99.40
sdj	0.00	0.00	2275.00	0.00	291200.00	0.00	256.00	1.80	0.79	0.79	0.00	0.44	99.30
sdj	0.00	0.00	2328.00	0.00	297984.00	0.00	256.00	1.78	0.77	0.77	0.00	0.43	99.60

Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	r_await	w_await	svctm	%util
sdj	0.00	0.00	2313.00	0.00	296064.00	0.00	256.00	1.81	0.78	0.78	0.00	0.43	99.10
sdj	0.00	0.00	2269.00	0.00	290432.00	0.00	256.00	1.81	0.80	0.80	0.00	0.44	99.80
sdj	0.00	0.00	2215.00	0.00	283520.00	0.00	256.00	1.79	0.81	0.81	0.00	0.44	98.10

dd if=/dev/sdj of=/dev/null iflag=direct bs=4096k

Device:	rrqm/s	wrqm/s	r/s	w/s	rkB/s	wkB/s	avgrq-sz	avgqu-sz	await	r_await	w_await	svctm	%util
sdj	0.00	0.00	1118.00	0.00	572416.00	0.00	1024.00	5.29	4.73	4.73	0.00	0.88	97.90
sdj	0.00	0.00	1085.00	0.00	555520.00	0.00	1024.00	4.88	4.49	4.49	0.00	0.90	97.50
sdj	0.00	0.00	1103.00	0.00	564736.00	0.00	1024.00	5.10	4.64	4.64	0.00	0.88	97.20
sdj	0.00	0.00	1128.00	0.00	577536.00	0.00	1024.00	5.35	4.75	4.75	0.00	0.87	97.70
sdj	0.00	0.00	1153.00	0.00	590336.00	0.00	1024.00	4.88	4.24	4.24	0.00	0.84	97.40

Промежуточные результаты

	None (Read)	ZLIB	LZ4
Default	290 MB/s	85 MB/s	~240 MB/s
Direct I/O	682 MB/s	94 MB/s	~370 MB/s

Замена алгоритма закрывает проблемы резервного копирования дисков с малым service time.

Что делать с дисками с большим service time?



Промежуточные результаты

Операция	Время на 1GB, секунд
Read (default)	3.5
Read (Direct I/O)	1.5
Compression (GZIP)	8
Compression (LZ4)	1
Ceph HDD Read	**30 (30 MB/s)**
Ceph SSD Read	**5 (200MB/s)**



Новая архитектура

Делегирование драйверу доступа к диску

Многопроцессная и многопоточная реализация

POSIX shared memory

Новый формат

Многопоточное сжатие



Сломать фреймворк

```
def backup(self, backup_info, volume_info):
    file_descriptor = self.open_volume(volume_info)
    try:
        self.driver.backup(backup_info, file_descriptor)
    finally:
        self.close_volume(volume_info, file_descriptor)
```

```
def backup(self, backup_info, volume_info):
    if self.driver.version() >= 2:
        source = volume_info
    else:
        source = self.open_volume(volume_info)

    try:
        self.backup(backup_info, source)
    finally:
        if self.driver.version() < 2:
            self.close_volume(volume_info, source)
```

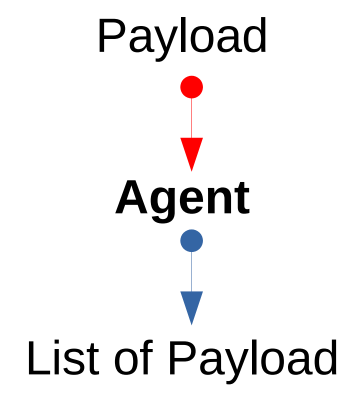


Многопроцессная архитектура

session_agent

- **Agent 1**
- **Agent 2**

  …
- **Agent N**



{
  “size”: 4096
}

[
  { “offset”: 0, “size”: 1024” },
  { “offset”: 1024, “size”: 1024” },
  { “offset”: 2048, “size”: 1024” },
  { “offset”: 3072, “size”: 1024” }
]



Многопроцессная архитектура

Backup

- **Master**
- **Reader**
- **Checksum**
- **DiffHandler**
- **Compressor**
- **DataWriter**
- **MetadataWriter**



Многопроцессная архитектура

Backup

- Master
- Reader
- Checksum
- DiffHandler
- Compressor
- DataWriter
- MetadataWriter

Restore

- **Master**
- **BackupReader**
- **Decompressor**
- **Writer**



Многопроцессная архитектура

Backup

- Master
- Reader
- Checksum
- DiffHandler
- Compressor
- DataWriter
- MetadataWriter

Restore

- Master
- BackupReader
- Decompressor
- Writer

Delete

- S3Worker



Zero-copy между агентами

POSIX IPC shared memory

Backup master

{ “offset”: 0, “size”: 4096}

Data reader

{ “offset”: 0, “size”: 4096, “shm_id”: “reader_12345_1”}

Checksum

{ “offset”: 0, “size”: 4096,
“shm_id”: “reader_12345_1”,
“hash”: “abc1234567890def” }

Compressor

{ “offset”: 0, “size”: 4096,
“shm_id”: “cmpt_12345_1”,
“hash”: “abc1234567890def” }



Асинхронный процессинг

{ “offset”: 0, “size”: 1024 }

{ “offset”: 1024, “size”: 1024 }

{ “offset”: 2048, “size”: 1024 }

{ “offset”: 3072, “size”: 1024 }

{ “offset”: 0, “size”: 1024, “shm_id”: “d1” }

{ “offset”: 2048, “size”: 1024, “shm_id”: “d2” }

{ “offset”: 3072, “size”: 1024, “shm_id”: “d3” }
{ “offset”: 1024, “size”: 1024, “shm_id”: “d4” }

Даёт очень существенный прирост для тонких волюмов …

Если data consumer успеет обработать “сгенерированные” данные

Поддержка инкрементальных копий

{ “offset”: 0, “size”: 1024 }

{ }

{ “offset”: 1024, “size”: 1024 }

{ “offset”: 1024, “size”: 1024, “object_offset”: 40, “compressed_size”: 1024 }

{ “offset”: 2048, “size”: 1024 }

{ }

{ “offset”: 3072, “size”: 1024 }

{ “offset”: 3072, “size”: 1024, “object_offset”: 0, “compressed_size”: 40 }

unchanged

unchanged

Изменение формата

{ “offset”: 0, “size”: 1024, “object_offset”: 32, “compressed_size”: 128 }

{ “offset”: 1024, “size”: 1024, “object_offset”: 200, “compressed_size”: 1024 }

{ “offset”: 2048, “size”: 1024, “object_offset”: 0, “compressed_size”: 32 }

{ “offset”: 3072, “size”: 1024, “object_offset”: 160, “compressed_size”: 40 }



“Сборка мусора”

```
{ “offset”: 0, “size”: 4096”,
  “shm_id”: “reader_12345_1” }


[
  { “offset”: 0, “size”: 1024”,
    “compressed_size”: 512,
    “shm_id”: “cmpt_12345_1”
   },
   { “offset”: 1024, “size”: 1024”,
     “compressed_size”: 512,
     “shm_id”: “cmpt_12345_2”
   }
]
```

In resources:

- shm: reader_12345_1

Out resources:

- shm: cmpt_12345_1
- shm: cmpt_12345_2

Resources to free:

- **shm: reader_12345_1**



“Сборка мусора”

```
{ “offset”: 0, “size”: 4096”,
  “shm_id”: “reader_12345_1” }


[
  { “offset”: 0, “size”: 1024”,
    “compressed_size”: 512,
    “shm_id”: “reader_12345_1”
   },
   { “offset”: 1024, “size”: 1024”,
     “compressed_size”: 512,
     “shm_id”: “cmpt_12345_2”
   }
]
```

In resources:

- shm: reader_12345_1

Out resources:

- shm: reader_12345_1
- shm: cmpt_12345_2

Nothing to free



“Сборка мусора”

```python
def handle_message(self, input, output, resource_manager):
    ...
    shm = resource_manager.create_shm(4096)
    ...
    output.append(Payload(shm_id=shm.id))
```

Запрос ресурсов через менеджер?

- Ресурсы учетены
- Будут освобождены
- Могут быть переиспользованы
- Можно переделать механизм аллокации



Как можно меньше зависимостей

- POSIX IPC shared memory
- UNIX domain socket
- JSON
- Сложные зависимости вынесены в отдельные агенты

Результат:

Возможность протестировать работу бэкапа без развертывания дополнительных сервисов – используется при проверке инфраструктуры

Или бэкапить не только OpenStack



Новые характеристики

До 1.5GB/секунду на один диск

Поддержка L1

40GB/сек виртуального пространства

~18GB/сек использованного пространства

Цена: большой расход памяти:

1GB/TB контрольные суммы
2GB/TB метаданные
2GB/TB сериализация



Загрузка сети

C10 = 18 in / 5.6 out
C19 = 25 in / 4.8 out
C24 = 30 in / 4.8 out
C25 = 25 in / 5.3 out
C26 = 20 in / 7.9 out
C27 = 25 in / 5.8 out

Итого

143 Gb/s чтение
34 Gb/s запись



Заметные доработки

Прямой доступ к снапшотам

Механизмы шедулинга с учетом зон доступности

Отказ от шедулинга “восстановление через тот же хост”



Интересные факты

Почти все отказы вызваны отказом системы чьи диски копируют (Guest Agent)

Единичные сбои вызваны отказами инфраструктуры или оборудования

Сбоев собственно разработанного драйвера после запуска в эксплуатацию почти не отмечалось

Объем тестирующего кода ~ в 2 раза больше чем объем кода драйвера



Вместо заключения

Стандартные механизмы I/O могут заметно снизить производительность на “однонаправленых” нагрузках (~100% запись или ~100% чтение)

Фреймворковый подход может накладывать ограничения и обходить их будет “больно” – но это существенно выгоднее, чем втискиваться в них

Высокопроизводительные решения требуют распараллеливания нагрузки. Распараллеливание требует соответствующих форматов данных.

